Finding the Most Similar Concepts in two Different Ontologies

Adolfo Guzman-Arenas, Jesus M. Olivares-Ceja

Centro de Investigación en Computación, Instituto Politécnico Nacional
07738 Mexico City, MEXICO
{a.guzman,jesuso}@acm.org

Abstract. A concise manner to send information from agent A to B is to use phrases constructed with the *concepts* of A: to use the *concepts* as the atomic tokens to be transmitted. Unfortunately, tokens from A are not understood by (they do not map into) the ontology of B, since in general each ontology has its own address space. Instead, A and B need to use a *common communication language*, such as English: the transmission tokens are English words.

An algorithm is presented that finds the concept c_B in O_B (the ontology of B) most closely resembling a given concept c_A. That is, given a concept from ontology O_A, a method is provided to find the *most similar concept* in O_B, as well as the similarity *sim* between both concepts. Examples are given.

1 Introduction and objectives

How can we communicate our *concepts*, what we really mean? Two persons (or agents) A and B can communicate through previously agreed stereotypes, such as the *calling sequence* between a caller program and a called subroutine. This requires previous agreement between A and B. This paper deals with communication with little previous consensus: A and B agree only to share a given *communication language.* The purpose of the communication is for A and for B to fulfill its objectives or goals. That is, we shall define a successful communication if A and B are closer to their goals as the result of such communication.

What can an agent do to meaningfully communicate with other agents (or persons), even when they had not made any very specific comitment to share a private ontology and communication protocol? Concept communication can not be fulfilled through direct exchange of concepts belonging to an ontology, since *they do not share* the same ontology. Instead, communication should be sought through a common language. Lucky agents can agree on a language whose words have a *unique meaning.* Others need to use an ambiguous language (such as a natural language) to share knowledge. This gives rise to imperfect understanding and confusion. This is the trust of this paper.

The objective of this work is to find the most similar (in meaning) object in B's ontology corresponding to a given object in A's ontology, and to measure their

similarity. Example: Assume A wants to transmit its concept `grapefruit`[1] to B. To this end, A translates it into word grapefruit, which is then transmitted to B. But B has no such word in its ontology. Thus, B asks A "what is a grapefruit?" A answers "it is a citric" (by seeing that `citric` is the father of `grapefruit` in O_A). Unfortunately, B has no concept to map word "citric". So B asks A "what is a citric?" A answers "it is a fruit". Now, O_B has concept `fruit` denoted by word fruit. But $\texttt{fruit}_B$ (the subindex B means "in O_B") has several children: B knows several fruits. Now B has to determine wich of the children of $\texttt{fruit}_B$ most resembles $\texttt{grapefruit}_A$. It may do so by seeing which child of $\texttt{fruit}_B$ has children quite similar to those children of $\texttt{grapefruit}_A$. Or by seeing which fruits in O_B have skin, bone, weight... similar to those of $\texttt{grapefruit}_A$. Unfortunately, the problem is recursive: what is `skin` for B is `epidermis` for A, and `peel` for C. $\texttt{weight}_A$ is in kilograms, whereas $\texttt{weight}_B$ is in pounds. So the comparison has to continue recursively. §2 gives a precise description of the algorithm.

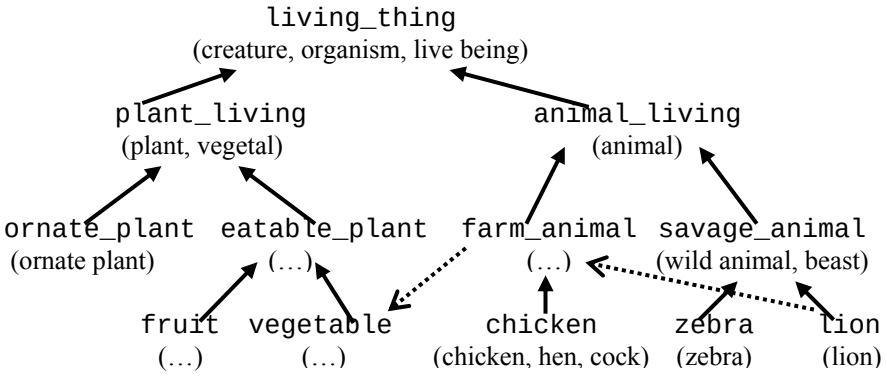

Figure 1. An ontology consists of a tree of concepts (nodes) under the *subset* relation (solid arrows), with other relations such as *eats* (dotted arrows), and with words associated to each concept (in parenthesis after each concept; some are omitted). Nodes also have (property, value) pairs, not shown in the figure

1.1 Ontologies

Knowledge is the concrete internalization of facts, attributes and relations among real-world entities ♦ It is stored as concepts; it is *measured* in "number of concepts."

Concept. An object, relation, property, action, idea, entity or thing that is well known to many people, so that it has a name: a word(s) in a natural language. ♦ Examples: `cat-chien`, `to_fly_in_air`, `angry_mad`. So, *concepts have names*: those words used to denote them. A concept is unambiguous, by definition. Unfortunately, the names given by different people to a concept differ and, more unfortunately, the same word is given to two concepts (examples: words mole;

[1] In this paper, concepts appear in `Courier font`.

star; can). Thus, *words are ambiguous*, [2] *while concepts are not.* A person or agent, when receiving words from some speaker, has to solve their ambiguity in order to understand the speaker, by mapping the words to the "right" concept in his/her/its own ontology. The mapping of words to concepts is called *disambiguation*.

If two agents do not share a concept, at least partially, they can not communicate it or about it. A concept has (property, value) pairs associated with it.

Ontology. It is a formal explicit specification of a shared conceptualization [5]. ♦ It is a hierarchy or taxonomy of the concepts we know.[3] We represent an ontology as a tree, where each node is a concept with directed arcs (representing the relation `subset` and, at the leaves, perhaps the relation `member_of` instead of `subset`) to other concepts. Other relations (such as `part_of`, `eats-ingests`, `lives_in`, ...) can be drawn, with arcs of different types (figure 1). In general, these relations are also nodes in another part of the ontology.

Associated words. To each concept (node) there are several English words[4] associated: those who denote it or have such concept as its meaning. Example: concept `mad_angry` has associated (is denoted by) words angry, crossed, pissed-of, mad, irritated, incensed. Example: Word mole denotes a `small_rodent`, a `spy_infiltrator` and also a `blemish_in_skin`.

1.2 Related work

[12] represents concepts in a simpler format, called a *hierarchy.* Most works (for instance [11]) on ontologies involve the construction of a single ontology, even those that do collaborative design [8]. Often, ontologies are built for man-machine interaction [10] and not for machine-machine interaction. [1] tries to identify conceptually similar documents, but uses a single ontology. [3, 4] do the same using a topic hierarchy: a kind of ontology. [9] seeks to communicate several agents sharing a single ontology. The authors have been motivated [6, 7] by the need of agents to communicate with unknown agents, so that not much *a priori* agreement between them is possible. With respect to concept comparison, an ancestor of our COM (§2, appears first in [13]) matching mechanism is [2], based on the theory of analogy.

2 Most similar concepts in two different ontologies

The most similar concept c_B in O_B to concept c_A in O_A is found by the COM algorithm using the function *sim*(c_A) (called "*hallar* (c_A)" in [13]) as described in the four cases below. It considers a concept, its parents and sons. In this section, for each case, a tree structure shows the situation and a snapshot of a screen presents an example. Assume that agent A emits (sends) to B words[5] corresponding to c_A, and

[2] Some symbols or words are unambiguous: 3, Abraham Lincoln, π, (30°N, 15°W).

[3] Each concept that I know and has a name is shared, since it was named by somebody else.

[4] Or word phrases, such as "domestic animal".

[5] Remember, an agent can not send a node to another agent, just words denoting it.

also sends words corresponding to the father of c_A, denoted by p_A. COM finds c_B = *sim*(c_A), the concept in O_B most similar to c_A. *sim* also returns a similarity value *sv,* a number between 0 and 1 denoting how similar such returned concept c_B was to c_A.

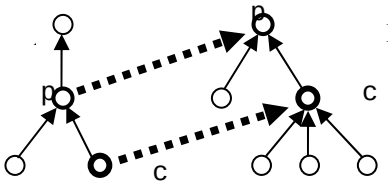

Figure 2. Case (a). Words from c_A and p_A match words from c_B and p_B

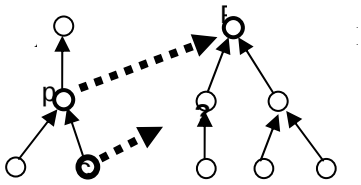

Figure 3. Case (b). Words from p_A match words from p_B but c_A has no equivalence

Case a) We look in O_B for two nodes p_B and c_B, such that: (1) the words associated to c_B coincide with most of the words (received by B from A)[6] of c_A; and (2) the words associated to p_B coincide with most of the words[6] corresponding to p_A; and (3) p_B is the father, grandfather or great-grandfather[7] of c_B.

If such p_B and c_B are found, then c_B is the nearest concept to c_A; the answer is c_B and the algorithm finishes returning *sv* = 1. Figure 2 represents this situation. Figure 4 shows the screenshot of COM when seeking in B the concept most similar to $\texttt{apple}_A$. The answer is concept $\texttt{apple}_B$ in B with *sv = 1*.

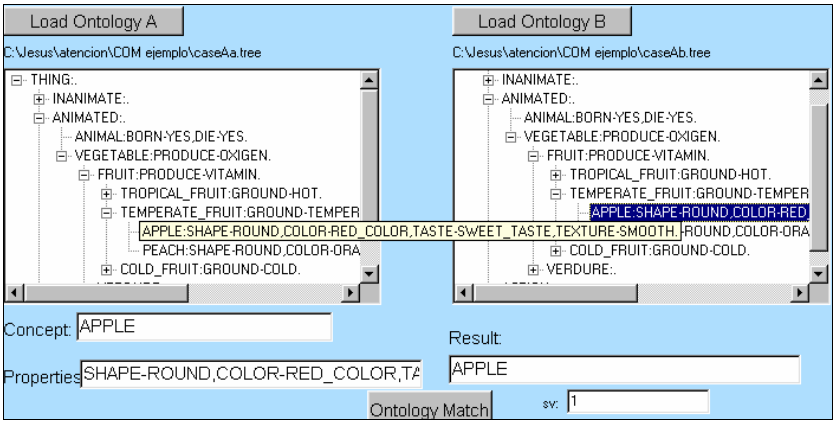

Figure 4. Case (a). Screen with the execution of COM for the case shown in Fig. 2

[6] We have found useful the threshold 0.5: more than half of the compared entities must coincide.

[7] If p_B is found more than three levels up, the "semantic distance" is too high and *sim* says "no match."

Case b) This case occurs when (2) of case (a) holds, but (1) and (3) do not. p_B is found in O_B but c_B is not. See Figure 3. In this case, *sim* (which Olivares calls *hallar*) is called recursively, and we try to compute $p_B' = sim(p_A)$ to confirm that p_B is the ancestor of concept of interest (c_A).

(1) If the p_B' found is `thing`, the root of O_B, the algorithm returns `not_found` and concludes; *sv* = 0;

(2) Otherwise, a special child of p_B, to be called c_B', is searched in O_B, such that:

A. Most[6] of the pairs (property, value) of c_B' coincide with the corresponding pairs of c_A. Children of p_B with just a few matching properties[6] or values are rejected. If the candidate c_B' analyzed has children, they are checked (using *sim* recursively) for a reasonable match[6] with the children of c_A. If a c_B' is found with the desired properties, the algorithm reports success returning c_B' as the concept in O_B most similar to c_A. *sv* = the fraction of pairs of c_B' coinciding with corresponding pairs of c_A.

B. Otherwise c_B' is sought among the sons of the father (in B) of p_B; that is, among the brothers of p_B; if necessary, among the sons of the sons of p_B; that is, among the grandsons of p_B. If found, the answer is c_B'. *sv* = the *sv* returned by c_B' multiplied by 0.8 if c_B' was found among the sons of p_B,[8] or by $0.8^2 = 0.64$ if found among the grandsons of p_B.

C. If such c_B' is not found, then the node nearest to c_A is some son of p_B, therefore *sim* returns the remark (`son_of` p_B) and the algorithm concludes. *sv* = 0.5 (an arbitrary but reasonable value). For example, if A sends words that correspond to the pair (c_A = `kiwi`, p_A = `fruit`), and B has the concept `fruit` but doesn't have the concept `kiwi` nor any similar `fruit`, in this case, the concept `kiwi` (of A) is translated by B into `(son_of fruit)`, which means "some `fruit` I don't know" or "some `fruit` I do not have in my ontology."

Figure 5 shows the execution of COM for case (b)2(A). In this case concept $\texttt{kiwi}_A$ has no equivalent in B. Here $\texttt{rare\_fruit}_B$ is chosen from B as the most similar concept because parents coincide and properties of $\texttt{kiwi}_A$ and $\texttt{rare\_fruit}_B$ are similar (that was calculated using COM recursively for each property-value). *sv = 0.8* because the exact equivalent concept in B was not found.

Case c) This case occurs when (1) of case (a) holds but (2) and (3) do not. See figure 6. c_B is found but p_B is not. We try to ascertain whether the grandfather (in O_B) of c_B has words that match[6] those of p_A (corresponding words that are equal exceed 50%), or if the great-grandfather of c_B in O_B has such matching[6] words.

(1) If that is the case, the concept in O_B more similar to p_A is the grandfather (or the great-grandfather) of c_B, and the algorithm finishes returning c_B. *sv* = 0.8 for the grandfather case, and 0.8^2 for the great-grandfather case.

[8] We have found that 0.8 allows for a fast decay as one moves up from father to grandfather and up.

(2) Otherwise (parents do not match), we verify two conditions:
   A. Most[6] of the properties (and their corresponding values) of c_B should coincide (using *sim*) with those of c_A ; and
   B. Most of the children of c_A should coincide (using *sim*) with most[6] of the children of c_B.

   If the properties in (A) and the children in (B) coincide, the algorithm concludes with response c_B, although it did not find in O_B the p_B that corresponds to the concept p_A in O_A. *sv* = the fraction of properties and children of c_B matching with corresponding entities of c_A.
(3) If even fewer properties and children are *sim*ilar then response is `(probably` c_B`)` and the algorithm finishes. *sv* is computed like in (2)B.
(4) If neither properties nor children are *sim*ilar, response is `not_found` and the algorithm finishes. *sv* = 0.

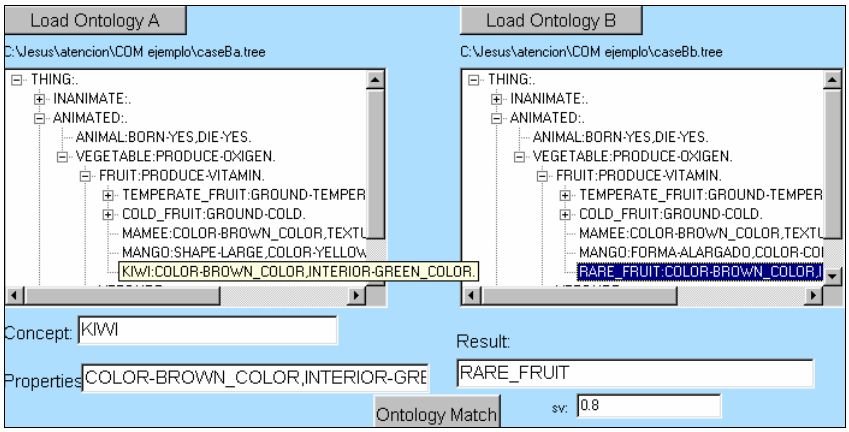

Figure 5. Case (b). Screen with the execution of COM corresponding to Figure 3

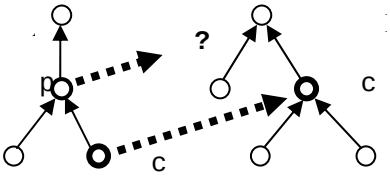

Figure 6. Case (c). Words from c_A match with words of c_B but there is no equivalence for words of p_A. See Figure 8

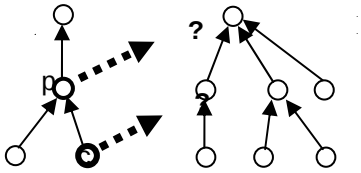

Figure 7. Case (d). There are no words from c_A nor p_A that match with words of B

Figure 8 shows an example of case (c)(2). In this case we use COM to seek in B the most similar concept to $\texttt{apple}_A$. Here concepts match but parents do not ($\texttt{fruit}_A$, $\texttt{food}_B$) (words are different for each parent), therefore similarity of the properties are used (calling recursively to COM). *sv* = 0.8 because parents do not coincide.

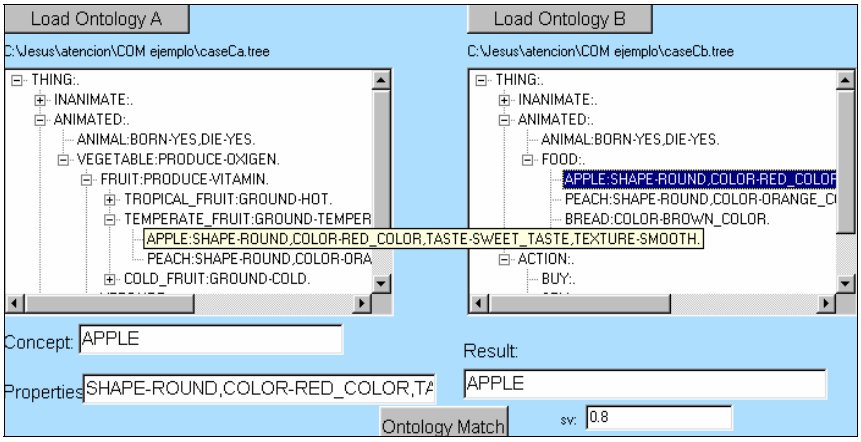

Figure 8. Case (c). Screen with the execution of COM corresponding to figure 6

Case d) If neither c_B nor p_B are found, the algorithm concludes returning the response `not_found`. *sv* = 0. c_A could not find a similar node in O_B. The agents may have different ontologies (they know about different subjects) or they do not share a common communication language. See figures 7 and 9.

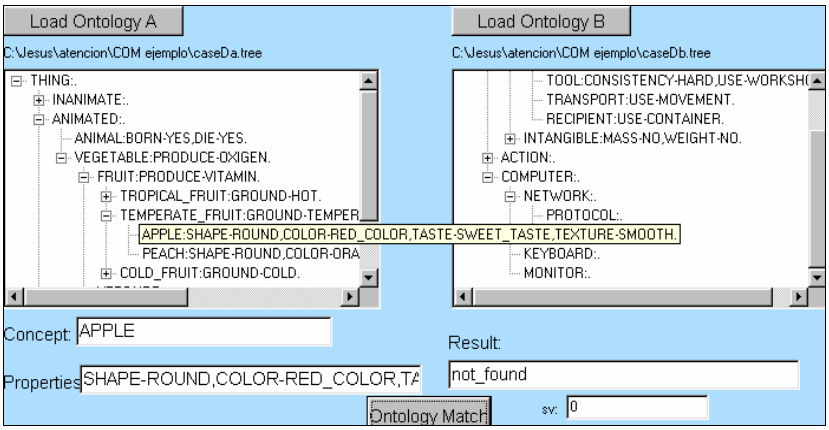

Figure 9. Case (d). Screen with the execution of COM for case (d). Ontologies refer mostly to different areas. COM returns `not_found` with *sv = 0*

Figure 9 shows the execution of case (d). Observe that ontology O_A is mainly about fruits while O_B is mainly about Computer Science. There are some concepts in common, but not the involved concepts. *sv = 0*.

sim is not symmetric. If c_B is the concept most similar to c_A, it is not necessarily true that c_A is the concept most similar to c_B. Example: O_A knows ten kinds of

$\texttt{hammer}_A$, while O_B only knows $\texttt{hammer}_B$ (a general hammer). Then, COM maps each of the ten $\texttt{hammer}_A$ into $\texttt{hammer}_B$, while $\texttt{hammer}_B$ best maps into, say, $\texttt{hammer\_for\_carpenter}_A$ [12].

The function *sim* is only defined between a concept c_A in O_A and *the most similar concept* c_B in O_B.

Ontology A

```
thing:.
    living_creature:.
        animal:.
        plant_living:color-green_color,produce-
oxigen.
            melon:.
            bean:.
    tool:.
        screwdriver:.
        key_tool:.
    data:.
        field:.
            integer_field:.
            real_field:.
            double_field:.
            float_field:.
        key_data:.
            foreign_key:.
            Primary_key:.
```

Figure 10. Ontology A. Used to compute similarity to concepts in ontology B

2.1 Examples of similarity

Now we give examples for *sim,* the similarity between two concepts, each from one ontology. Here we assume that properties like relations and colors are part of both ontologies. For simplicity properties are shown only where needed. Properties appear after the colon as relation-value pairs. For ontologies A and B (Figures 10 and 11):

sim($\texttt{field}_A$) = $\texttt{field}_B$ with *sv = 1* because words of concepts and parents coincide. This is an example of case (a).

sim($\texttt{key\_tool}_A$) = $\texttt{key\_tool}_B$ with *sv = 1.* This is an example of case(a), where words of the parent and concept in A match words of corresponding nodes in B. Although word 'key' denotes (belongs to the *associated words* of) both concepts $\texttt{key\_data}_B$ and $\texttt{key\_tool}_B$, the words of $\texttt{tool}_A$ only map into those of $\texttt{tool}_B$ and $\texttt{key\_tool}_B$ is selected without ambiguity.

sim($\texttt{screwdriver}_A$) = $\texttt{(son\_of tool}_B\texttt{)}$ with *sv = 0.5.* This is case (b): parents coincide, but in ontology B there is no concept similar to $\texttt{screwdriver}_A$, therefore the algorithm detects that agent A is referring to a son of concept $\texttt{tool}_B$.

sim(plant_living$_A$) = vegetable$_B$ with *sv = 0.8.* This an example of case (b) when parents coincide but the concepts do not. In this case properties of concepts are used to establish the similarity among concepts. The similarity of the properties is calculated using the COM recursively for each property and value.

sim(double_field$_A$) = not_found and *sv = 0*. This is an example of case (d) when no concept nor parent are found in B. The ontology A has sent B a concept of which B has no idea.

sim(melon$_A$) = not_found and *sv = 0*. This is other example of case (d) where words sent to B from A do not match a pair parent-concept in B.

Ontology B

```
thing:.
    living_creature:.
        animal:.
        vegetable:color-green_color,produce-oxigen.
            apple:.
            bean:.
    tool:.
        hammer:.
        key_tool:.
    data:.
        field:.
        key_data:.
```

Figure 11. Ontology B. Used to compute similarity to concepts in ontology A

2.2 Conclusions

Methods embodied in a computer program are given to allow concept exchange and understanding between agents with different ontologies, so that there is no need to agree first on a standard set of concept definitions. Given a concept, a procedure for finding the most similar concept in another ontology is shown. The procedure also finds a measure of the similarity *sv* between concepts c_A and c_B. Our methods need further testing against large, vastly different, or practical ontologies.

In contrast, most efforts to communicate two agents take one of these approaches:

1. The same person or programmer writes (generates) both agents, so that pre-established *ad hoc* communicating sequences ("calling sequences," with predefined order of arguments and their meaning) are possible. This approach, of course, will fail if an agent is trying to communicate with agents built by somebody else.
2. Agents use a common or "standard" ontology to exchange information. This is the approach taken by CYC [11]. Standard ontologies are difficult and slow to build (they have to be designed by committee, most likely). Another deficiency: since new concepts appear each day, they slowly trickle to the standard ontology, so that it always stays behind current knowledge.

Even for approach (2), a language to convey other entities built out of concepts: complex objects (which do not have a name), actions, desires, plans, algorithms… (not just concepts) is needed. Such language is beyond this paper; hints of it at [13].

Our approach allows communication in spite of different ontologies, and needs neither (1) nor (2).

Acknowledgments. Work herein reported was partially supported by NSF-CONACYT Grant 32973-A and Project CGPI-IPN 18.07 (20010778). Olivares received a PIFI research assistantship from CGPI-IPN. Guzman-Arenas has a SNI *National Scientist* Award from SNI-CONACYT.

References

1. John Everett, D Bobrow, R Stolle, R Crouch, V de Paiva, C Condoravdi, M van den Berg, and L Polyani. (2002) Making ontologies work for resolving redundancies across documents. *Communication of the ACM* **45,** 2, 55-60. February.
2. K. Forbus, B. Falkenhainer, D. Gentner. (1989) The structure mapping engine: algorithms and examples. *Artificial Intelligence* **41**, 1, 1-63.
3. A. Gelbukh, G. Sidorov, and A. Guzman-Arenas. (1999) Use of a weighted document topic hierarchy for document classification. *Text, Speech, Dialogue*, 133-138. Pilsen, Chech Republic, September 13-17.
4. A. Gelbukh, G. Sidorov, and A. Guzman-Arenas. (1999) Document comparison with a weighted topic hierarchy. *DEXA-99*, 10th International Conference on Database and Expert System applications, Workshop on Document Analysis and Understanding for Document Databases, 566-570. Florence, Italy, August 30 to September 3.
5. Thomas R. Gruber (1993) Toward Principles for the Design of Ontologies Used for Knowledge Sharing, in *Formal Ontology in Conceptual Analysis and Knowledge Representation*, Nicola Guarino and Roberto Poli (eds.), Kluwer Academic Publishers.
6. A. Guzman, Jesus Olivares, Araceli Demetrio and Carmen Dominguez, (2000) Interaction of purposeful agents that use different ontologies. *Lecture Notes in Artificial Intelligence* (*LNAI)* **1793**, 557-573. Osvaldo Cairo, Enrique Sucar, F. J. Cantu (eds). Springer Verlag.
7. A. Guzman, C. Dominguez, and J. Olivares. (2002) Reacting to unexpected events and communicating in spite of mixed ontologies In *LNAI* **2313**, 377-386.
8. Cloyde W. Holsapple and K. D. Joshi. (2002) A collaborative approach to ontology design. *Comm. ACM* **45**, 2, 42-47. February.
9. M. N. Huhns; M. P. Singh. and T. Ksiezyk (1997) Global Information Management Via Local Autonomous Agents. In *Readings in Agents,* M. N. Huhns, Munindar P. Singh, (eds.). Morgan Kauffmann Publishers, Inc. San Francisco, CA
10. Henry Kim. (2002) Predicting how ontologies for the semantic web will evolve. *Comm. ACM* **45**, 2, 48-54. February.
11. Douglas B. Lenat, R. V. Guha, Karen Pittman, Dexter Pratt and Mary Shepherd (1990) Cyc: Toward Programs with Common Sense, *Comm. ACM* **33**, 9, 30 – 49.
12. Serguei Levachkine, A. Guzman-Arenas (2002) Hierarchy as a new data type for qualitative variables. Submitted to *Data and Knowledge Engineering.*
13. Jesus Olivares (2002) *An Interaction Model among Purposeful Agents, Mixed Ontologies and Unexpected Events*. Ph. D. Thesis, CIC-IPN. In Spanish. Available on line at http:// www.jesusolivares.com/interaction/publica